Build tools basics – Maven & Gradle. Git

Working with Git - repositories, commands, branches, pull requests, resolving conflicts

Agenda for This Session

- Ant

- Maven

- Gradle

- Git - repositories, commands, branches, pull requests, resolving conflicts

Apache Ant

- Apache Ant is a software tool for automating software build processes, which originated from the Apache Tomcat project in early 2000.

- Replacement for the Make build tool of Unix, created due to a number of problems with Unix's make.

- Similar to Make but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects.

- With Make, actions required to create a target are specified as shell commands which are specific to the platform on which runs.

- Ant solves this problem by providing a large amount of built-in functionality designed to behave similarly on all platforms.

Apache Ant - I

```xml
<?xml version="1.0"?>

<project name="Invoicing with Views" default="compile">

    <presetdef name="javac"><javac includeantruntime="false" /></presetdef>

    <property name="build.dir" location="${basedir}/bin"/>

    <target name="clean" description="remove intermediate files">

        <delete dir="${build.dir}"/>
    </target>

    <target name="clobber" depends="clean" description="remove all artifact files">

        <delete file="invoicing.jar"/>
    </target>

    <target name="compile" description="compile the Java source code to class files">

        <mkdir dir="${build.dir}"/>

        <javac srcdir="." destdir="${build.dir}"/>
    </target>
    ...
```

Apache Ant - II

```xml
...
    <target name="jar" depends="compile" description="create a Jar file for the app">
        <jar destfile="invoicing.jar">
            <fileset dir="${build.dir}" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="invoicing.view.MainMenu"/>
            </manifest>
        </jar>
    </target>
    <target name="run" depends="compile" description="run the application">
        <java classname="invoicing.view.MainMenu"
              classpath="${java.class.path};${build.dir}" dir="." fork="true" />
    </target>
    <target name="runJar" depends="jar" description="run the app from the jar file">
        <java jar="invoicing.jar" dir="." failonerror="true" fork="true" />
    </target>
</project>
```

Build Integration with Maven

What is Maven?

- What is Maven Apache Maven is a software project management and comprehension tool. Based on the concept of a **Project Object Model (POM)**, Maven can manage a project's build, reporting and documentation from a central piece of configuration – **pom.xml**

- Advantages over Ant:

  – Eliminate the hassle of maintaining complicated scripts

  – All the functionality required to build your project , i.e. , clean, compile, copy resources, install, deploy etc. is built right into the Maven

  – Cross Project Reuse – Ant has no convenient way to reuse target across projects, But Maven does provide this functionality

Maven Main Phases and Commands

Although hardly a comprehensive list, these are the most common default lifecycle phases excuted:

- **validate**: validate the project is correct and all necessary information is available
- **compile**: compile the source code of the project
- **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test**: process and deploy the package if necessary into an environment where integration tests can be run
- **verify**: run any checks to verify the package is valid and meets quality criteria
- **install**: install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.
- There are two other Maven lifecycles of note beyond the default list above. They are
- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

Phase → **Plugin:Goal**

- *process-resources* -> resources:resources

- *compile* -> compiler:compile

- *process-test-resources* -> resources:testResources

- *test-compile* -> compiler:testCompile

- *test* -> surefire:test

- *package* -> ejb:ejb OR ejb3:ejb3 OR jar:jar OR par:par OR rar:rar OR war:war

- *install* -> install:install

- *deploy* -> deploy:deploy

9

Maven Common Plugins
https://www.baeldung.com/core-maven-plugins

- Resources Plugin – copies resources

- Compiler Plugin - compile the source and test code

- Failsafe Plugin – run tests with cleanup

- Surefire Plugin – run tests

- Verifier Plugin – check invariants

- Install Plugin – install to local Maven repo

- Deploy Plugin – deploy to remote Maven repo

- Site Plugin – run Maven info site (site:site, site:run)

- Clean Plugin – cleans generated dirs (mvn clean)

Maven Dependency Management

- Apache Maven – https://spring.io/guides/gs/maven/

- Common arguments: `mvn compile, mvn package,     mvn clean dependency:copy-dependencies package, mvn install, mvn clean deploy site-deploy,`

- `mvn -q compile exec:java -Dexec.mainClass= "simpledemo.AnnotationSpringDIDemo"`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.iproduct.spring</groupId>
    <artifactId>01-introduction-maven</artifactId>
    <version>1.0-SNAPSHOT</version>
```

11

Maven Configuration (continued)

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.0.5.RELEASE</version>
    </dependency>
</dependencies>

<repositories>
    <repository>
        <id>io.spring.repo.maven.release</id>
        <url>http://repo.spring.io/release/</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

Maven Configuration (continued)

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>9</source>
                <target>9</target>
            </configuration>
        </plugin>
    </plugins>
</build>

</project>
```

Maven Configuration (enhanced)

```xml
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>5.0.5.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
</dependencies>
```

14

Gradle Dependency Management

- Gradle – https://spring.io/guides/gs/gradle/

- Init new project/ convert exisitng from Maven: **gradle init**

- Build project: **gradle build**

- Build project: **gradle run**

- Example configuration:

```
group 'org.iproduct.spring'
version '1.0-SNAPSHOT'
apply plugin: 'java'
apply plugin: 'application'
mainClassName = 'org.iproduct.spring.demo.xmlconfig.HelloWorldSpringDI'

sourceCompatibility = 1.8
```
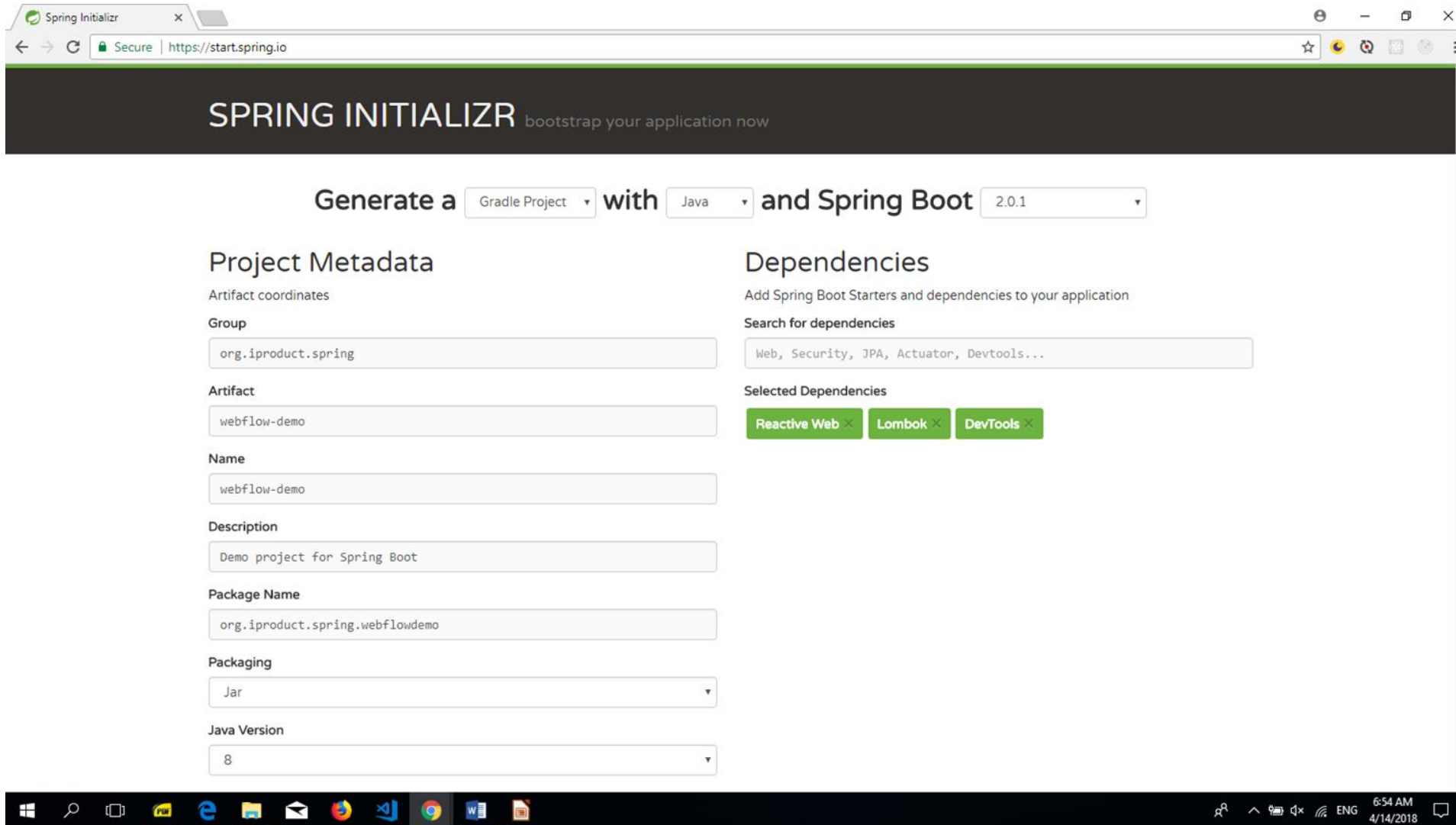
Gradle Configuration (continued)

```
task runApp(type: JavaExec) {
    classpath = sourceSets.main.runtimeClasspath
    main = 'org.iproduct.spring.demo.xmlconfig.HelloWorldSpringDI'
}

repositories {
    mavenLocal()
    mavenCentral()
    maven { url "https://repo.spring.io/snapshot" }
    maven { url "https://repo.spring.io/milestone" }
}

dependencies {
    implementation group: 'org.springframework',
            name: 'spring-context', version: '5.0.5.RELEASE'
    testImplementation group: 'junit', name: 'junit', version: '4.12'
}
```

Java Library plugin configurations for dependencies:

Configuration name	Role	Description
api	API dependencies	Dependencies which are transitively exported to consumers, for compile time and runtime.
implementation	implementation dependencies	Dependencies which are purely internal and not meant to be exposed to consumers (they are still exposed to consumers at runtime).
compileOnly	compile only dependencies	Dependencies which are required at compile time, but not at runtime. This typically includes dependencies which are shaded when found at runtime.
compileOnlyApi	compile only API dependencies	Dependencies which are required at compile time by your module and consumers, but not at runtime. This typically includes dependencies which are shaded when found at runtime.
runtimeOnly	runtime dependencies	Dependencies which are only required at runtime, not at compile time.
testImplementation	test dependencies	Dependencies used to compile tests.
testCompileOnly	test compile only dependencies	Dependencies which are only required at test compile time, but should not leak into the runtime. This typically includes dependencies which are shaded when found at runtime.
testRuntimeOnly	test runtime dependencies	Dependencies which are only required at test runtime, and not at test compile time.

Making Projects Easy: Spring Boot 2

Resources

- Guide to the Core Maven Plugins –
  https://www.baeldung.com/core-maven-plugins

- Guide to the Core Maven PluginsGradle User Manual –
  https://docs.gradle.org/current/userguide/userguide.html

- Pro Git book – https://git-scm.com/book/en/v2

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD